UNIVERSIDAD TORCUATO DI TELLA

[Versión: 3 de noviembre de 2022]

Trabajo Práctico - Algoritmos y Estructuras de Datos

Licenciatura en Tecnologías Digitales, UTDT

Segundo semestre 2022

- El TP se debe realizar en grupos de 3 personas.
- La fecha de entrega es hasta el domingo 27 de noviembre inclusive.
- Se evaluará no solo la correctitud técnica de la solución propuesta sino también la claridad del código escrito.

Descripción del problema

Se desea implementar un tipo de datos `Comunidad` que modela un sistema de registro de empresas. Cada empresa está identificada con un string `empresa` y un identificador `id` que son únicos para esa empresa (no hay dos empresas con el mismo `id` o el mismo string `empresa`). Dos empresas pueden asociarse entre sí para realizar proyectos en conjunto. Diremos que una empresa es "popular" si está asociada con al menos otras 5 empresas.

Consigna

1. Definir una estructura de representación en el archivo `Comunidad.h` que permite satisfacer los **requerimientos de complejidad**.
2. Escribir, en español, como comentario en `Comunidad.h` las condiciones que debe cumplir la estructura para ser válida (el invariante de representación)
   - Dar un ejemplo de valores para la estructura que cumpla el invariante.
   - Dar un ejemplo de valores para la estructura que **NO** cumpla el invariante.
3. Escribir en el archivo `Comunidad.cpp` la implementación de los métodos respetando los **requerimientos de complejidad**. No está permitido modificar la interfaz pública de la clase.

Sugerencias para la estructura de representación:

- Utilizar clases provistas por la *Biblioteca Estándar* de C++, aprovechando sus órdenes de complejidad.
- No es necesario diseñar estructuras manejando memoria dinámica de manera explícita.

Interfaz de la clase

```
class Comunidad{
  public:
    Comunidad();

    const set<int> & inscriptos() const;
    string obtener_empresa(int id) const;
    const set<string> & obtener_socios(int id) const;
    int cantidad_asociaciones() const;
    int obtener_id(string empresa) const;
    bool es_empresa_popular(string empresa) const;

    void inscribir(string empresa, int id);
    void desinscribir(int id);
    void asociar(string empresa_A, string empresa_B);
    void desasociar(string empresa_A, string empresa_B);

  private:
    /* ... */
};
```

Notar que los métodos `inscriptos()` y `obtener_socios(int id)` devuelven un contenedor *por referencia*. Esto significa que al momento en que la función devuelve el contenedor no se computa ningun costo de copiarlo.

Requerimientos de complejidad

`Comunidad()`	$O(1)$
`inscriptos()`	$O(1)$
`obtener_empresa(int id)`	$O(\log n)$
`obtener_socios(int id)`	$O(\log n)$
`cantidad_asociaciones()`	$O(1)$
`obtener_id(string empresa)`	sin requerimiento
`es_empresa_popular(string empresa)`	$O(1)$ *promedio*.
`inscribir(string empresa, int id)`	$O(\log n)$
`desinscribir(int id)`	sin requerimiento
`asociar(string empresa_A, string empresa_B)`	sin requerimiento
`desasociar(string empresa_A, string empresa_B)`	sin requerimiento

El tamaño n es la cantidad de empresas registradas en la `Comunidad`. Para los requerimientos de complejidad pueden asumir que el costo de copiar el nombre de una empresa (los strings `empresa`, `empresa_A`, `empresa_B`) es constante O(1).

Descripción detallada de las operaciones

- `Comunidad();`
  Pre: Verdadero
  Post: Construye una comunidad vacía.

- `const set<int> & inscriptos() const`
  Pre: Verdadero
  Post: Devuelve *por referencia* el conjunto de identificadores (`id`) de todas las empresas actualmente inscriptas.

- `string obtener_empresa(int id) const`
  Pre: `id` está en el conjunto `inscriptos()`
  Post: Devuelve el nombre de la empresa correspondiente al identificador `id`.

- `const set<string> & obtener_socios(int id) const`
  Pre: `id` está en el conjunto `inscriptos()`
  Post: Devuelve *por referencia* el conjunto de los nombres de empresas asociadas con la empresa `id`.

- `int cantidad_asociaciones() const`
  Pre: Verdadero
  Post: Devuelve la cantidad total de relaciones de sociedad actualmente en la comunidad. Si dos empresas A y B son socias eso cuenta como una asociacion.

- `void inscribir(string empresa, int id)`
  Pre: `id` no está en el conjunto `inscriptos()`, `empresa` no corresponde a ningún `id` inscripto.
  Post: El conjunto `inscriptos()` tiene un elemento más: `id`; y `obtener_empresa(id)` devuelve `empresa`.

- `void desinscribir(int id)`
  Pre: `id` está en el conjunto `inscriptos()`
  Post: Se elimina la empresa `id` del sistema y se eliminan todas las asociaciones de las que participaba.

- `void asociar(string empresa_A, string empresa_B)`
  Pre: `empresa_A` y `empresa_B` son distintos y, además, están en la comunidad.
  Post: El conjunto `obtener_socios(empresa_A)` tiene un elemento más: `empresa_B` y el conjunto `obtener_socios(empresa_B)` tiene un elemento más: `empresa_A`.

- `void desasociar(string empresa_A, string empresa_B)`
  Pre: `empresa_A` y `empresa_B` son distintos y, además, están en la comunidad y están asociadas.
  Post: El conjunto `obtener_socios(empresa_A)` tiene un elemento menos: `empresa_B`. El conjunto `obtener_socios(empresa_B)` tiene un elemento menos: `empresa_A`.

- `int obtener_id(string empresa) const`
  Pre: `empresa` está en la comunidad.
  Post: Devuelve el `id` de `inscriptos()` al que le corresponde el nombre `empresa`.

- `bool es_empresa_popular(string empresa) const`
  Pre: `empresa` está en la comunidad.
  Post: Indica si la cantidad de socios de `empresa` es al menos 5.